RL-TR-95-288
Final Technical Report
January 1996

REAL-TIME PARALLEL SOFTWARE DESIGN CASE STUDY: IMPLEMENTATION OF THE RASSP SAR BENCHMARK ON THE INTEL PARAGON

The MITRE Corporation

Curtis P. Brown, Richard A. Games, and John J. Vaccaro

Rome Laboratory
Air Force Materiel Command
Rome, New York

RL-TR-95-288 has been reviewed and is approved for publication.

APPROVED: [illegible]

RALPH KOHLER
Project Engineer

FOR THE COMMANDER: Gary D. Barmore

GARY D. BARMORE, Major, USAF
Deputy Director of Surveillance & Photonics

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE January 1996	3. REPORT TYPE AND DATES COVERED Final Oct 94 - Sep 95

4. TITLE AND SUBTITLE REAL-TIME PARALLEL SOFTWARE DESIGN CASE STUDY: IMPLEMENTATION OF THE RASSP SAR BENCHMARK ON THE INTEL PARAGON	5. FUNDING NUMBERS C - F19628-94-C-0001 PE - 62702F PR - MOIE TA - 74 WU - 11
6. AUTHOR(S) Curtis P. Brown, Richard A. Games, and John J. Vaccaro	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) The MITRE Corporation 202 Burlington Road Bedford MA 01730-1420	8. PERFORMING ORGANIZATION REPORT NUMBER N/A
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Rome Laboratory/OCSS 26 Electronic Pky Rome NY 13441-4514	10. SPONSORING/MONITORING AGENCY REPORT NUMBER RL-TR-95-288

11. SUPPLEMENTARY NOTES

Rome Laboratory Project Engineer: Ralph Kohler/OCSS/(315) 330-2016

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

A software design process for mapping real-time applications onto massively parallel processors is described. The design methodology incorporates a software test bench used to evaluate the level of real-time performance the processing nodes are capable of delivering. The test bench results incorporate the overheads of periodic processing (simple communication, control flow, buffering, etc.) and therefore provide realistic performance levels. The final integration step maintains the simple test bench interfaces and reduces the complexity of integrating the components to satisfy the timing requirements of the overall application. The process is applied to implement the RASSP SAR benchmark on an Intel Paragon. The initial implementation uses 12 Paragon GP nodes for a single polarization. Under OSF/1, this 12 node configuration satisfies all the real-time requirements. Under SUNMOS, a streamlined high performance operating system available on the Paragon, the throughput improves significantly with sustained processor utilization approaching 40%. A projected implementation of the RASSP SAR benchmark on the Embedded Touchstone suggests that all three polarizations can be processed (including I/O) using 14 out of its 16 MP nodes.

14. SUBJECT TERMS Real-time applications, Synthetic operative radar, SAR, Paragon, SUNMOS	15. NUMBER OF PAGES 33
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102

ACKNOWLEDGMENTS

This work was supported by the United States Air Force Electronic Systems Center and performed under MITRE Mission Oriented Investigation and Experimentation (MOIE) Project 74260 of contract F19628-94-C-0001, managed by Rome Laboratory/OCSS. This work was supported in part by a grant of HPC time from the DOD HPC Major Shared Resource Center, Wright-Patterson Air Force Base, on an Intel Paragon^{TM}. We wish to thank the Honeywell Corporation, Space Systems, Clearwater, Florida, and Sandia National Laboratory for the use of their Paragons. Paragon is a trademark of the Intel Corporation. All other product names, trademarks, and registered trademarks are the property of their respective holders.

The authors would also like to acknowledge Mark Flanzbaum who provided high performance library calls and other assistance.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

SECTION 1

INTRODUCTION

High performance computing can play a significant role in the military's evolving seamless design methodology to rapidly produce systems with reduced life-cycle costs by providing homogeneous communication and scalable computing frameworks. Two related programs underway at ARPA are focusing on providing efficient and effective computing solutions for the Department of Defense. The Rapid Prototyping of Application Specific Signal Processors (RASSP) program, sponsored by the Electronic Systems Technology Office (ESTO), has a goal of improving the process of specifying, designing, manufacturing, and supporting complex digital signal processors. On the other hand, the Computing Systems Technology Office (CSTO) has initiated the Embedded Computing program whose goal is to make available commercial scalable high performance computing technology to real-time embedded applications. For example, the CSTO/Honeywell Embedded Touchstone project is packaging the commercial Intel Paragon massively parallel processor (MPP). Our implementation of the RASSP synthetic aperture radar (SAR) image formation benchmark on the Intel Paragon explores the boundary between these two ARPA programs [Brown, et al., 1995].

The transition of commercial programmable MPPs into real-time embedded applications promises to have a major impact on future military computing. However target applications will utilize MPPs quite differently than has historically been the case in the scientific computing community where real-time, size, weight and power constraints do not exist. Embedded real-time applications require high sustained processing rates, high sustained message passing rates, real-time services at the processing nodes, and real-time internode communication services. One particular aspect of the work reported here is the establishment of a scalable software design and benchmarking methodology for real-time embedded applications that replaces the traditional benchmark approaches and notions of scalability drawn from scientific computing.

After providing a brief introduction to the Intel Paragon and Embedded Touchstone, this paper focuses on a software design process for mapping real-time applications onto MPPs. The design methodology incorporates a software test bench used to evaluate the level of real-time performance the processing nodes are capable of delivering. The test bench results incorporate the overheads of periodic processing (simple communication, control flow, buffering, etc.) and therefore provide realistic performance levels. The final integration step maintains the simple test bench interfaces and dedicates additional resources to deal with more complicated global communications. This makes the original single node test bench results highly predictive and reduces the complexity of integrating the components to satisfy the timing requirements of the overall application.

The process is applied to the RASSP SAR benchmark to obtain an initial implementation that uses 12 Paragon GP nodes (each with a single i860XP microprocessor for computation) for a single polarization. Under OSF/1 this 12 node configuration satisfies all the real-time requirements. Under SUNMOS, a streamlined high performance operating system available on the Paragon, the throughput improves significantly with sustained processor utilization approaching 40%. The test bench provides significant insight into the current level of real-time guarantees available from existing non-real time operating systems. A projected implementation of the RASSP SAR benchmark on the Embedded Touchstone suggests that all three polarizations can be processed (including I/O) using 14 out of its 16 MP nodes. Finally, real-time scalability for the SAR application is discussed.

SECTION 2

INTEL PARAGON AND THE EMBEDDED TOUCHSTONE

The Intel Paragon is a multiple instruction multiple data (MIMD) distributed memory MPP with nodes used for either service, computing, or I/O. Service nodes primarily compile, link and start jobs. Compute nodes are typically not time-shared and are used exclusively to execute application code. Nodes dedicated to I/O may have interfaces to RAID, a parallel file system, Ethernet, or HiPPI. The current generation GP node has 32 Megabytes (MB) of DRAM and two 50 MHz i860XP microprocessors; the next generation MP node has 3 i860XPs and 64 MB of DRAM.

The Paragon is delivered with a commercial version of the OSF/1 operating system. OSF/1 is a full featured micro-kernel-based operating system that provides system services such as virtual memory, threads and inter-process communication (IPC), and networking support for ftp and NFS [Zajcew, et al., 1993]. OSF/1 consumes up to 12 Mbytes of memory including the micro-kernel, OSF/1 AD server and system buffer space. Under OSF/1, one of the i860XPs at each node is dedicated to message handling and is unavailable to the application. A streamlined alternative operating system called SUNMOS (approximately 250 kbytes) has been developed by Sandia National Laboratory and the University of New Mexico [Maccabe et al., 1993]. SUNMOS provides minimal services but delivers higher performance (lower latency and higher throughput) and provides the application programmer access to the second i860XP at each node. Neither operating system has a preemptible kernel, which is a requirement for a real-time operating system, creating particular challenges in applying the current Intel Paragon for embedded real-time applications.

The nodes are interconnected in a two-dimensional mesh topology with routers at each node employing dimension-order wormhole routing. The nodes have a direct memory access interface to the mesh through a single ported network interface chip. The mesh has a theoretical capacity of 200 MB/s with OSF/1, revision 1.2.7, achieving a sustained communication throughput of 80 MB/s and SUNMOS, B-step revision 1.6.3, sustaining 160 MB/s. Intel supports the NX application programming interface (API) for message passing, and a subset of the API is available under SUNMOS [McCurley, 1993]. Applications which adhere to the subset are source compatible between OSF/1 and SUNMOS.

ARPA/CSTO has sponsored the Honeywell Embedded Touchstone program [Blitzer, 1993] to develop an embeddable version of the Intel Paragon containing MP-nodes with three i860XPs per node. A requirement of this packaging effort is that the Embedded Touchstone will execute the exact same software as the corresponding commercial system. Honeywell is developing a 16 node prototype (4.8 Gflop/s) that will occupy .5 cu. ft. and consume 560 watts of average prime power. Honeywell is currently working in collaboration with the Open Software Foundation-Research Institute (OSF-RI) on a real-time operating system that will run on the Embedded Touchstone. The physical parameters of the Embedded

Touchstone make it an attractive target architecture for the RASSP benchmark. Assessing the limitation due to the current non real-time operating system is one objective of this work.

SECTION 3

REAL-TIME PARALLEL SOFTWARE DESIGN METHODOLOGY

All engineering design endeavors are facilitated by a clear specification of the system requirements. Real-time embedded systems are characterized by functional, timing and physical requirements. Military systems have additional requirements based on life-cycle costs which encompass maintainability, reliability, supportability, extensibility and scalability. This section describes a software design approach that utilizes a test bench implemented on the target architecture as a means of assuring functional and timing correctness. A small example is provided, with a more comprehensive example (the RASSP benchmark) to follow in Section 4.

3.1 SOFTWARE DESIGN PROCESS

The process starts with clear specifications of the functional and timing requirements and a high-level model of the target architecture. The specifications and model are used to perform standard data and control flow analysis that serves to identify the major computation and communication kernels of the application. The functional analysis includes memory requirements and operation counts that combine with timing specifications to yield operation throughput requirements. The throughput requirements and a knowledge of the system's capabilities can be used to get a rough estimate of the number of compute nodes that will be required for each task. A real-time test bench (described in the next section) is used to establish the actual level of real-time performance the system is capable of delivering on each of the identified kernels. After functional correctness is established, perhaps using specified ground truth data, the test bench facilitates the process of tuning the kernel to meet the timing requirements. This may be accomplished by including vendor supplied library functions or even customized assembly language functions. If the timing requirements at a node cannot be satisfied, then the algorithm partition needs to be reevaluated.

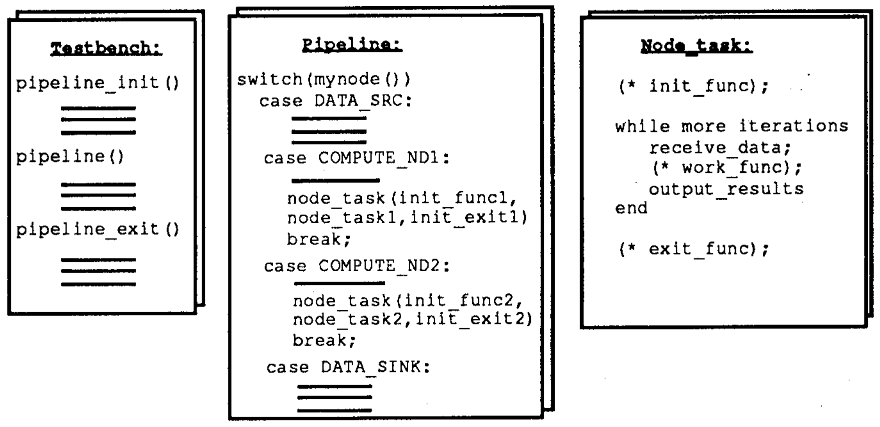
After timing requirements at the node level have been satisfied, the same test bench is used to exercise collections of nodes as they are integrated, culminating with a complete system test. Maintaining the test bench interfaces in the final integration step usually guarantees timing compliance when nodes are integrated. This requires the use of dedicated processing resources to implement the more complicated global communication requirements found in applications. These additional nodes are primarily dedicated to packing and unpacking the messages involved in these global communication patterns, e.g., the "distributed corner turn" in the current two-dimensional processing example. This means that the nodes doing the processing are relieved of this duty (these activities would alternatively take place in the application code running on the compute processor) and so can deliver more useful work. This in turn reduces the number of compute nodes required. The test bench also facilitates rapid node reconfiguration to allow swift system integration and to encourage experimental (perhaps ultimately adaptive) approaches to algorithmic partitioning and mapping for real-time embedded systems.

3.2 REAL-TIME TEST BENCH

The test bench assesses the level of real-time performance the system is capable of delivering [Brown, et al., 1994]. At the topmost level the test bench is a software template or "wrapper" that provides a data source and data sink between which the application is embedded. Currently the source and sink are each implemented on separate compute nodes of the MPP. An actual embedded system would stream data into an I/O node from an external real-time data source such as a sensor. The I/O node would buffer this incoming data and would correspond to the source node in the test bench. To avoid the excessive impact of file I/O, timing experiments are performed with limited data already resident and locked on the data source node; however, tests for functional correctness are supported through file access options on the data source node.

The test bench also provides a wrapper for the individual application kernels. The functions of real-time periodic processing such as buffering and flow control are incorporated into this second wrapper. This makes the performance results produced by the test bench very predictive as long as the test bench interfaces are maintained. The test bench is request driven and the application wrapper provides single and double buffered input options. The request driven characteristic of the test bench assures that "receives" for large data blocks are posted, and thus do not incur an extra memory copy from the kernel space to the application. Our experience has shown that eliminating unnecessary memory copies is essential for achieving high performance [Brown, et al., 1994].

Figure 3-1 shows pseudo-code for a test bench with two application nodes. The outer test bench wrapper consists of three functions: `pipeline_init`, `pipeline`, and `pipeline_exit`. The `pipeline_init` routine is used to configure the application under test. A node's initialization includes communication information (input source node(s), output destination node(s), and message sizes) as well as computation information (whether a computation can be done in place, single or double buffering, computation parameters such as FFT sizes). The `pipeline` routine is simply a switch on the logical node number which invokes an application wrapper with the appropriate work function. The `pipeline_exit` routine is used to do any cleanup that is required, such as testing functional correctness, outputting performance statistics, etc.



```
Testbench:
pipeline_init()
pipeline()
pipeline_exit()
```

```
Pipeline:
switch(mynode())
  case DATA_SRC:

  case COMPUTE_ND1:
    node_task(init_func1,
    node_task1,init_exit1)
    break;
  case COMPUTE_ND2:
    node_task(init_func2,
    node_task2,init_exit2)
    break;
  case DATA_SINK:
```

```
Node task:
(* init_func);

while more iterations
   receive_data;
   (* work_func);
   output_results
end

(* exit_func);
```

Figure 3-1. Test Bench Pseudo Code

The application wrapper is contained in the `pipeline` routine and similarly consists of an initialization function, a work function, and an exit function to implement the application compute/communication kernel. The initialization function is used to set up any initial state required, for example generating the twiddle factors for an FFT or reading in convolution kernels from a data file. The actual application kernel code resides in the work function, which is called from the application wrapper. On a given node the application wrapper establishes (through requests) the input data for the compute task, calls the work function through a function pointer, and satisfies downstream data requests with the generated application output. The exit function performs any required clean-up tasks.

The test bench is instrumented to timestamp three events at each node: (1) data receive, (2) compute, and (3) data send. The state transition events are timestamped at each specified node providing complete visibility into the parallel execution process. Graphical tools developed for viewing the timestamp data include timing diagrams, frame-to-frame work distribution profiles, and generation of probability density functions of the state durations. Worst case periods are used to guarantee compliance with real-time requirements, with the level of confidence in the design directly correlated to the duration of the experiment. Beyond these worst case times it is the actual distribution functions that fully characterize the performance. In all cases times can be converted to Mflop/s to reveal sustained processor utilization as measured as a percentage of peak Mflop/s.

3.3 TEST BENCH EXAMPLE

This section shows how the real-time test bench is used to evaluate the performance of a processing kernel under both OSF/1 and SUNMOS. The function under test in this example is a 1024 point fast convolution, which consists of a 1024-point complex FFT, a component-by-component multiplication by a 1024-point vector of complex weights, and a 1024-point inverse FFT. Vectors arrive in sequence at the processor to be individually convolved. A *granularity study* determines by how much the data should be blocked to allow more efficient coarse grain processing. If the real-time application had a strict latency requirement, then keeping the block size to a minimum would be preferred. In this example the test bench is implemented on three compute nodes of an Intel Paragon, one each for the data source, the processing node, and the data sink.

The minimum sustainable period is determined as a function of block size r. For each value of r, this minimum sustainable period corresponds to the largest time between successive data requests at the data sink as observed over a 30 second experiment. A fixed length experiment provides a level window into operating system activities and results in roughly the same number of total processed vectors independent of block size. The worst case minimum sustainable period is used to determine the sustained Mflop/s rate, where the total (single precision) flop count for the 1024-point convolution is taken as: $2(5 \times 1024 \times \log_2 1024) + 6 \times 1024$. Since the i860XP has a theoretical peak rate of 100 Mflop/s, this sustained rate corresponds to the percent utilization of the peak rate that is actually being sustained.

Figure 3-2 shows the results for OSF/1, revision 1.2.7. The block size corresponds to the number of rows along the x-axis. The Paragon OSF/1 operating system limits the granularity (block size r) at which efficient processing can be maintained. The reduction in sustained utilization for small block sizes results primarily from operating system dropouts—periodic preemptions by the operating system of the application thread executing the fast convolution code. These dropouts artificially limit the size of the minimum period especially for small block sizes when the durations of the worst case dropouts are comparable to the processing times required. There is also a decrease in communication efficiency for smaller block sizes, but this is a lower order effect in the case of OSF/1.

Figure 3-2 also illustrates the performance improvement if optimized library routines are incorporated. A completely portable C version of the fast convolution function was timed at approximately 8 Mflop/s (for blocks with $r = 40$, say). This improved to ~30 Mflop/s when the FFT and inverse FFT were replaced by optimized Kuck & Associates FFT calls. The C-coded vector component multiplication in the middle of these two FFT calls causes a significant performance degradation even though this glue code only represents 6% of the floating point operations. Incorporating an assembly-level vector-component-multiplication library call increases the performance to ~45 Mflop/s.

Figure 3-3 repeats the experiment under SUNMOS, B-step, revision 1.6.3, revealing significantly better and more predictable performance, primarily due to a combination of less severe operating system dropouts and reduced overhead in the communication system software. The best case now has sustained utilization at approximately 50%. These granularity studies reveal the trade-off that currently exists for the Paragon between minimizing latency (operating with small block sizes) and obtaining high utilization of the processing resources. This will be revisited in the RASSP benchmark discussion.

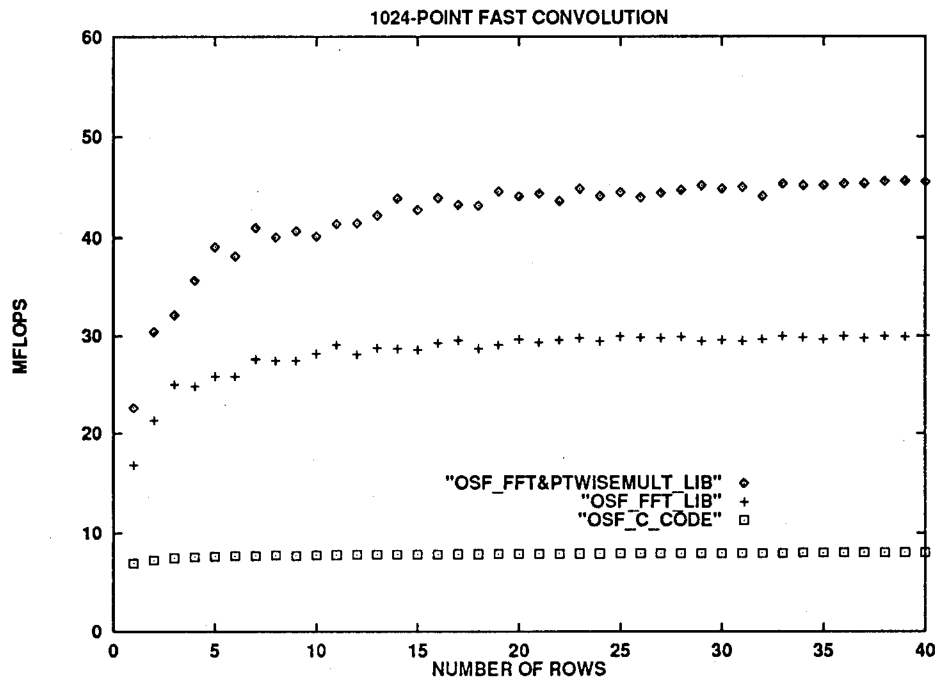


Figure 3-2. Fast Convolution Granularity Study for OSF/1, Revision 1.2.7

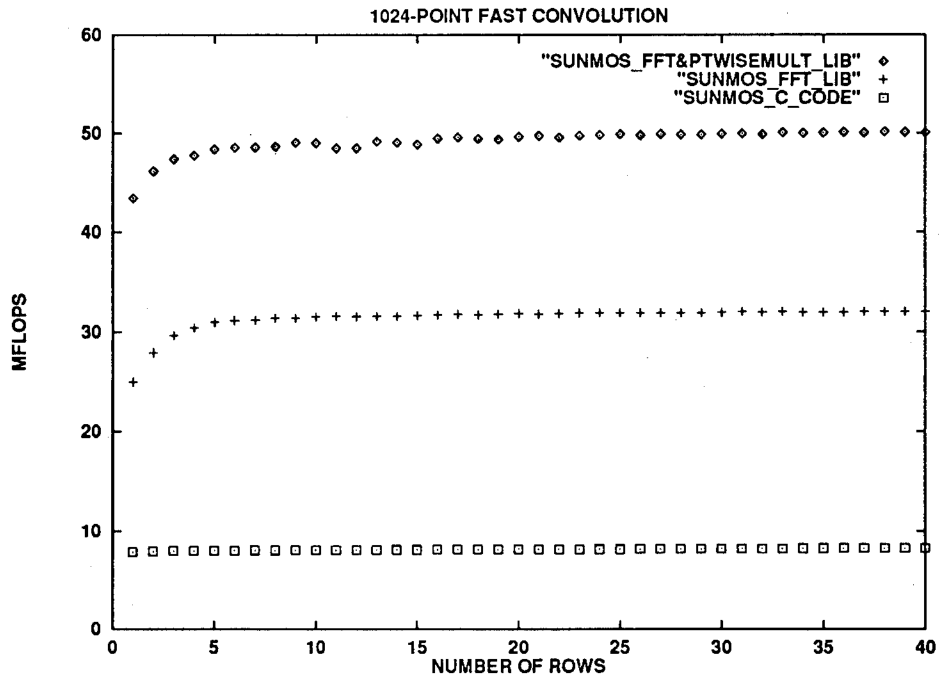


Figure 3-3. Fast Convolution Granularity Study for SUNMOS, B-step, Revision 1.6.3

Blank Pages

SECTION 4

REAL-TIME SYNTHETIC APERTURE RADAR IMPLEMENTATION

The ARPA sponsored Rapid Prototyping of Application Specific Signal Processors (RASSP) program is focused on improving the process of developing application specific digital signal processors. The effectiveness of the proposed process improvements is being evaluated with a series of benchmark exercises. The initial exercise is a synthetic aperture radar (SAR) image formation problem that has moderate computational throughput and memory requirements. The embedded requirements are consistent with real-time processing on board an unmanned air vehicle (UAV). This section describes the RASSP SAR benchmark and its implementation on the Intel Paragon. Projections are then made to a solution based on the Honeywell Embedded Touchstone. A brief discussion of real-time scalability for the SAR application is given.

4.1 RASSP SAR BENCHMARK

The RASSP SAR benchmark is described in [Zuerndorfer and Shaw, 1994]. It is based on processing radar data collected from the M.I.T. Lincoln Laboratory Advanced Detection Technology Sensor (ADTS). The ADTS is a Ka-band SAR sensor, navigation and data recording system hosted on a Gulf-stream G1 twin-engine aircraft. The radar's three polarization channels (HH, VV, and VH) can be processed independently. The benchmark is a stripmap SAR where the maximum pulse repetition frequency (PRF) of 556 Hz induces a minimum pulse-to-pulse period of 1.8 ms. One foot (0.3 m) resolution is obtained in both range and crossrange with a 375 m range swath nominally at a range of 7.26 km. The radar parameters and imaging geometry result in a benign SAR image formation process. Range migration less than one resolution cell results in separable range and crossrange processing.

Range processing is performed on individual radar pulse returns. Vectors of real video samples (a deramped linear FM range pulse) containing 4064 samples are first converted to 2024 baseband complex samples (video to baseband conversion). The in-phase (respectively, quadrature) samples are the result of filtering the even indexed (respectively, odd indexed) input samples. The two finite impulse response filters are conjugate-phase low pass filters (identical coefficients but in reverse order). Range compression is accomplished by performing a 2048 point FFT on this baseband vector. A real window is applied to the FFT input to control sidelobe levels. This input weight vector (window) incorporates a modulation by $(-1)^n$ to position the center of the range swath in the middle of the output vector. An additional window is applied after the FFT to compensate for radar cross section variations.

The separable nature of the ADTS SAR system allows the crossrange compression to be performed with a simple one-dimensional matched filter. This matched filter operates on individual range bins by processing across the range compressed radar pulses. The number of samples in the matched filter template (512) is determined by the antenna beamwidth and the PRF, and represents the number of radar pulses transmitted as the aircraft traverses the (at range) distance corresponding to the beamwidth. To take advantage of FFT efficiencies, the matched filter is implemented as an overlap-and-save fast convolution process [Oppenheim and Schafer, 1975] in which the transformed template coefficients are precomputed. The fast convolution operates on 1024 point windows to produce 512 valid output points.[1] Each of these output sections constitutes a frame of the stripmap image.

The RASSP timing requirements derive from the 556 Hz maximum PRF producing a 0.921 second/frame specification. This results in a 1.1 Gflop/s requirement to process all three polarizations. Latency is required to be less than three seconds. There is also an accuracy requirement that can be met with single precision floating point. The embedded requirements include a 60 pound payload and 500 watts of average prime power. This implies a 2 Mflop/s/watt solution.

4.2 REAL-TIME INTEL PARAGON IMPLEMENTATION

This section describes the application of the parallel software design process discussed in Section 3 to implement the RASSP SAR benchmark on the Intel Paragon. The functional and timing specifications are clearly stated in [Zuerndorfer and Shaw, 1994]. An executable specification in the form of sequential C code running on a SPARC workstation was also provided, along with ground-truth test data. Analysis of the SAR image formation algorithm begins by establishing the computational requirements of the three main computational kernels (1) video to baseband conversion, (2) range compression, and (3) azimuth compression. Figure 4-1 depicts the algorithmic operation for each of the kernels of the RASSP ADTS system.

[1] Although 513 valid output points are generated, only 512 points are used to maintain consistency with the RASSP ADTS benchmark description.

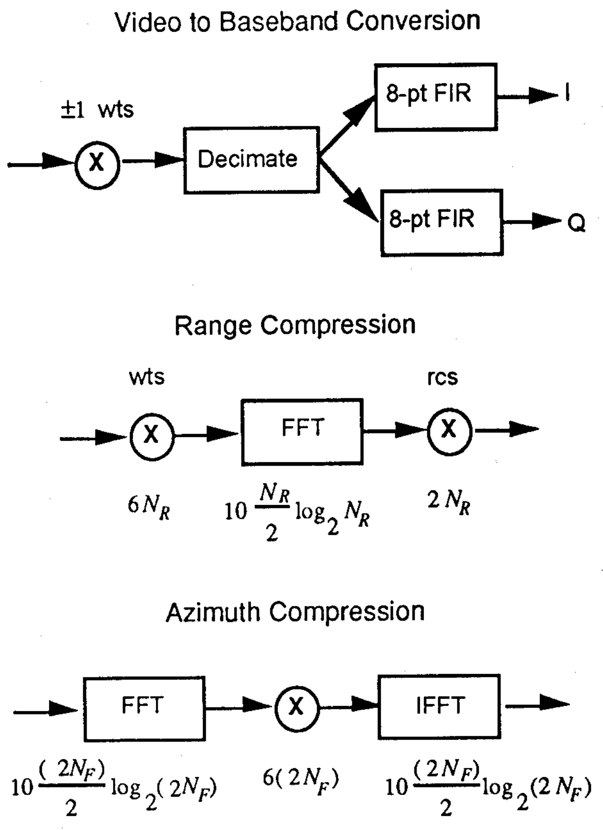

Video to Baseband Conversion
±1 wts
X
Decimate
8-pt FIR
I
8-pt FIR
Q
Range Compression
wts
rcs
X
FFT
X
$6N_R$
$10\frac{N_R}{2}\log_2 N_R$
$2N_R$
Azimuth Compression
FFT
X
IFFT
$10\frac{(2N_F)}{2}\log_2(2N_F)$
$6(2N_F)$
$10\frac{(2N_F)}{2}\log_2(2N_F)$

Figure 4-1. Kernel Functional Descriptions

The operations required for video to baseband conversion increase linearly with the number of pulses processed in each block, denoted by N_P :

$$\text{OPs} = N_P * 2024 * 2 * 8 * 2,$$

where 8-tap filters have been assumed. The required operation rate of 36 Mflop/s (based on the maximum PRF requirement of 556 Hz) is independent of N_P since the time available also scales linearly with number of pulses.

The operation count for the range compression FFT (assuming a complex pre-multiplier and a real post-multiplier) is

$$\text{OPs} = N_P[N_R(8 + 5\log_2 N_R)].$$

The operation rate is again invariant with block size N_P, and for an FFT size of $N_R = 2048$, the required performance is 72 Mflop/s.

The overlap and save fast convolution implementation of the azimuth compression results in transforms whose length (1024) is twice the frame size ($N_F = 512$). The operation count taken from Figure 4-1 is

$$\text{OPs} = N_R 2N_F\left[20N_F \log_2(2N_F) + 12N_F\right].$$

The resulting 241 Mflop/s throughput requirement is based on the 512 pulse frame period of 0.92 seconds. In order to make the azimuth processing as efficient as possible, a corner turn operation is required (to obtain contiguous samples in memory for fast convolution processing).

With the functional and timing requirements specified and the major computational and communication kernels identified, an initial pass at sizing can be made. A single node should suffice for the video to baseband conversion (36 Mflop/s). FFTs dominate the remaining computations. Realistically configured non repeated-instance FFTs (1024 or 2048 points) on the i860XP sustain 45–50 Mflop/s [Brown, et al., 1994]. The range compression requirement of 72 Mflop/s is unrealistic for a single node, but within reach for two nodes. Six nodes are expected to be required for the azimuth compression (241 Mflop/s).

Several alternative parallelization techniques exist (e.g., data parallel, pipelining, and round robin) and trade-offs are often driven by throughput and latency requirements. For instance, pipelining will improve throughput, but each stage of the pipeline will add latency. Although the data parallel approach is a low latency solution, it necessitates more complicated data flow.

The latency for the RASSP SAR benchmark is defined as the time between the arrival of the last pulse used to form the frame until the first corresponding image pixel is output. Latency is minimized by a front-end (video to baseband and range compression) solution that processes blocks with a small number N_P of pulses (fine grain), so that when the last pulse of the frame arrives, most of the other pulses for the frame have already been processed. Previously described granularity studies suggest lower bounds on N_P motivated by the desire to maintain adequate levels of processor utilization. In the current implementation, a blocksize (subframe) of $N_P = 64$ pulses was chosen; it is a factor of the frame size (512), and is well out onto the flat part of the performance curve.

The front end was implemented with a linear pipeline: the video to baseband and range compression functions each comprised a separate pipeline stage. The range compression was further parallelized using a "data-parallel pipelined" technique. Each range compression node passes the entire subframe of data but operates on only a fraction of it. This approach, although arguably less efficient, was motivated by a desire to keep the front-end pipeline linear thereby simplifying the corner turn operation. The azimuth compression back end is necessarily a frame based (coarse grain) process that is naturally implemented as a data-parallel process partitioned over range subswaths.

The test bench establishes the level of real-time performance the Paragon can actually deliver on the identified computation and communication kernels; see Figure 4-2. These results are similar to the granularity study described in Section 3.3, showing Mflop/s as a function of block size (number of rows). With some assembly level optimization, the video to baseband conversion can be accomplished (just barely) on a single node. Again the block size of N_P = 64 pulses is sufficiently large to accomplish this. Since each range compression node is getting approximately 25 Mflop/s (for sufficiently large blocksizes), a three-stage data-parallel pipeline is required to meet the specification of 72 Mflop/s. This appears to be too significant a cost for the simplified data flow yielded by the "data-parallel pipelined" technique; future scalable implementations will combine video to baseband and range compression functions on a single node and then utilize a data-parallel partition over the pulses in a frame. As expected, the azimuth compression required six nodes (a single node delivers approximately 45 Mflop/s.)

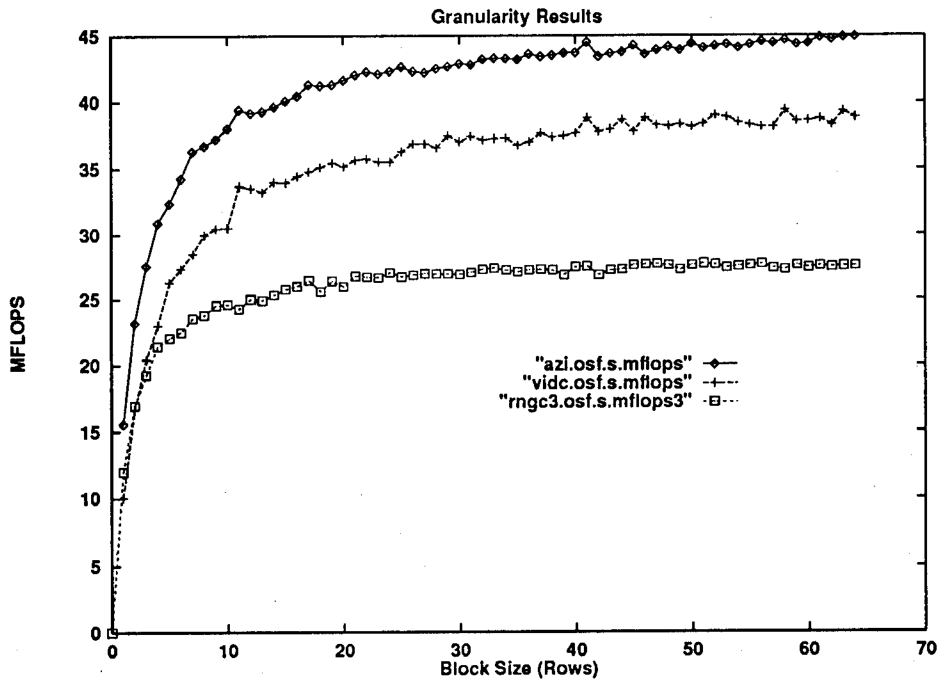


Figure 4-2. Granularity Results for SAR Application

Two particular constraints drove the design of the corner turn software: (1) OSF/1 consumes 12 of the available 32 MB of DRAM on each GP node requiring a two node solution to support buffering (a single overlap extended frame is 16 MB), and (2) the fine grained front end must be transitioned into the coarse grain back end. Since the front end is a simple linear pipeline (the benefit of the "data-parallel pipeline") the corner turn is not "distributed" (i.e., the two corner turn nodes do not have to exchange data). Each corner turn node is responsible for half of the range swath (corresponding to three azimuth nodes). The last range compression node sends its complete output to both corner turn nodes which operate on the appropriate data subset.

Each corner turn node has to accept eight subframes (64 pulses each) at a period of 0.115 seconds, transpose the pulses in each subframe, shift half of the old frame for the "overlap and save" operation, and output completely corner-turned frame data (512 pulses) to its three azimuth nodes. After the last subframe of the current frame has been transposed, the current frame is ready to be sent to the azimuth nodes. However, the first subframe of the next frame will arrive before the three sends can be completed. In the current implementation this output is delayed until after the first subframe of the next frame is buffered. Completed frame data is then sent to two out of the three azimuth nodes before the arrival of the second subframe. After the second subframe is buffered, the output to the third azimuth node is completed. There is also time to complete the "overlap and save" frame copy before the arrival and buffering of the third subframe. As a result, three subframes are buffered to allow for these output and frame copy operations. The remaining intervals between subframes are used to transpose buffered and newly arriving subframe data.

The single polarization solution using 12 GP nodes is shown in Figure 4-3. This solution is mapped to the mesh by a loader in a run-time library that is linked to the application code. In the current implementation each node receives the same program image and a switch statement in the application code determines the function performed based on the logical node number. The current mapping for a 4×4 mesh is shown in Figure 4-4. This mapping does not appear to have any serious communication contention problems. In the notation of Figure 4-4, corner turn node 0 sends data to azimuth nodes 0, 1, and 2; and corner turn node 1 sends data to azimuth nodes 3, 4, and 5. It is also possible to load different programs on the respective nodes, but this is usually avoided because of the increased complexity of the software configuration and maintenance.

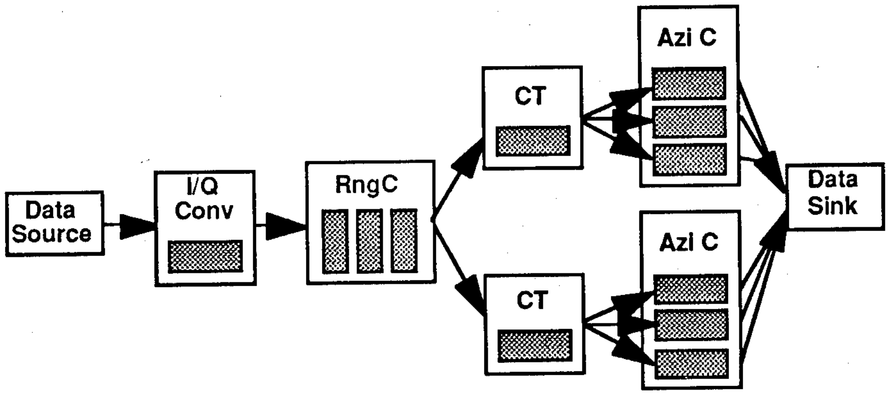


Figure 4-3. Single Polarization RASSP Benchmark Solution (GP Nodes)

Data Source	Video ---> I/Q	Range Compression 0	Range Compression 1
Data Sink			Range Compression 2
Azimuth Compression 2	Azimuth Compression 1	Azimuth Compression 0	Corner Turn 0
Azimuth Compression 5	Azimuth Compression 4	Azimuth Compression 3	Corner Turn 1

Figure 4-4. Mapping of the Single Polarization RASSP Benchmark Solution to a 4 x 4 Mesh (GP Nodes)

4.3 CURRENT PERFORMANCE RESULTS AND LEVEL OF EFFORT

Under OSF/1, revision 1.2.7, the 12 node single polarization implementation achieves a worst-case throughput performance of 0.85 second/frame and a 1.16 second latency, well within the real-time requirements of .92 second/frame and 3 seconds of latency. Under SUNMOS, B-step revision 1.6.5, the results improve to 0.71 second/frame throughput and a 1.02 second latency. These results are based on seven minute runs to account for the non real-time behavior of the operating systems. Figure 4-5 shows typical histograms of the time between successive arrivals of 1000 frames at the data sink under OSF/1 and SUNMOS. It is clear from Figure 4-5 that SUNMOS currently provides more predictable performance than OSF/1. The SUNMOS histogram, with a mean of 0.707 seconds, ranges between a minimum of .702 and a maximum of .712 seconds. The OSF/1 histogram, with a mean of 0.81 seconds, ranges between a minimum of .786 and a maximum of .835 seconds. The current SUNMOS results correspond to end-to-end processing utilization approaching 40% of the theoretically available 1200 Mflop/s of 12 GP nodes. Here we are assuming an OSF/1 compatible implementation that only has access to one compute processor per node. A SUNMOS-specific redesign with worst case performance closer to the specification and would reduce the number of processing nodes required.

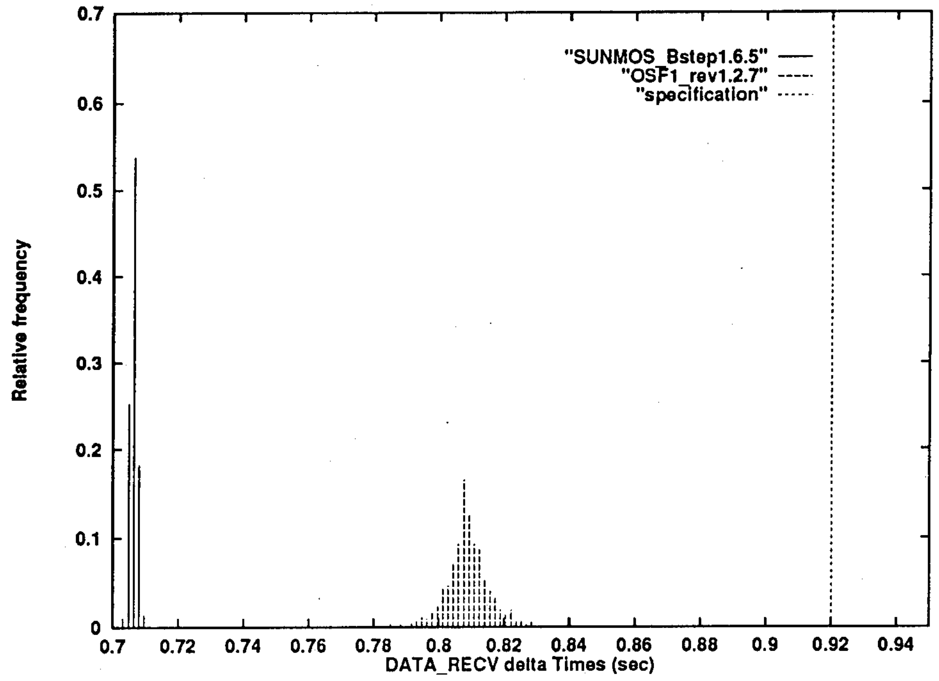


Figure 4-5. Histogram of the Periods for OSF/1 and SUNMOS

The implementation team had complementary skills and background. One staff member had a strong background in signal processing and SAR, but little experience with parallel machines. The other staff member had a strong background in programming parallel machines, but no background in signal processing or SAR.

The overall effort took 9.5 staff months, although 4 staff months were devoted to background, the development of the test bench and other supporting infrastructure. The implementation and design iteration of the application code itself took 2.5 staff months, including .5 staff months dedicated to developing optimized assembly-level code. Three staff months were spent testing and debugging. Debugging was more difficult than normal due to problems with system software on the Paragon and our use of remote machines. A more specific breakdown of activities is shown in Table 4-1.

Table 4-1. Software Development Level of Effort

Infrastructure: (4 staff-months)	
Background (ADTS/Paragon):	0.75 SM
Test bench coding:	1.50 SM
Generating results:	0.25 SM
Support/analysis tools:	1.50 SM
Application: (5.5 staff-months)	
Preliminary design partitions:	0.50 SM
Initial functional coding:	0.50 SM
Design iteration:	1.50 SM
Testing and debugging:	3.00 SM

4.4 EMBEDDED TOUCHSTONE PROJECTION

The next generation (MP) nodes of the Paragon have three i860XP microprocessors. If all three i860s are available to the application programmer (supported under SUNMOS), it is estimated that the RASSP SAR benchmark requirement could be met with 4 MP nodes per polarization, with 12 MP nodes for all three polarizations. Input could be accommodated by interfacing the RASSP fiber input source to a HiPPI I/O node. An additional I/O node could be required for output. With this estimate, the functional, physical, and timing requirements of the RASSP SAR benchmark would be satisfied with the Embedded Touchstone prototype using 14 out of its 16 MP nodes.

4.5 SCALABILITY FOR REAL-TIME

Parallel processing uses multiple processors to reduce the amount of time required to execute an algorithm, that is to *speed up* a computer program that implements the algorithm. The rate of reduction in the run time as additional processors are added is often regarded as a key descriptor of how effective parallel processing is for the particular application. A system's *scalability* refers to the nature of the progression of reduced run times as the number of processors is increased. For a fixed problem size, there is a limit to the speedup that can be obtained with some applications bottoming out quickly because they contain portions of inherently sequential operation (Amdahl's law). As a result, there have been alternative measures of scalability proposed that describe the behavior of the system as both problem size and the number of processors are increased [Gustafson]. The issue of scalability for real-time parallel processing is discussed in [Brown, et al., 1994].

A SAR problem scales in two ways. The simplest way is to increase the range swath with the same PRF. This does not effect the resolution but simply increases the number of samples within each pulse. A more aggressive scaling involves an improvement in resolution (resolution is assumed to be equal in range and azimuth) for a fixed range swath. In the range dimension the number of samples per pulse increases inversely proportional to the resolution reduction. In the azimuth dimension the wider beamwidth required increases both the azimuthal correlation template and the PRF so that the crossrange computation rate also increases inversely with resolution. With this second scaling, computational requirements increase as the square of the resolution reduction.

The present front end (video to baseband conversion and range compression) limits the scalability of the current implementation. The "data-parallel pipelined" range compression quickly loses efficiency as the number of stages increases. A scalable solution requires a data-parallel front end combining both video to baseband conversion and range compression functions on a single node. With some further optimization we anticipate such a data-parallel front end would eliminate two out of the four GP nodes used in the current implementation. A data-parallel front end will also necessitate a distributed corner turn (i.e., the corner turn nodes have to gather data from the front end, format messages, exchange data between themselves, unpack messages, and then scatter data to the back end).

A notional scalable SAR architecture is shown in Figure 4-6. One feature of this architecture is that it dedicates a separate stage to the distributed corner turn. In this way its overhead and impact on real-time scalability can be explicitly determined. Preliminary assessments of the scalability of the data exchange phase of the corner turn is given in [Brown, et al., 1994]. Scalable corner turn solutions are required for both scalable signal processing implementations and their synthesis with data flow shell tools. This is the subject of future work.

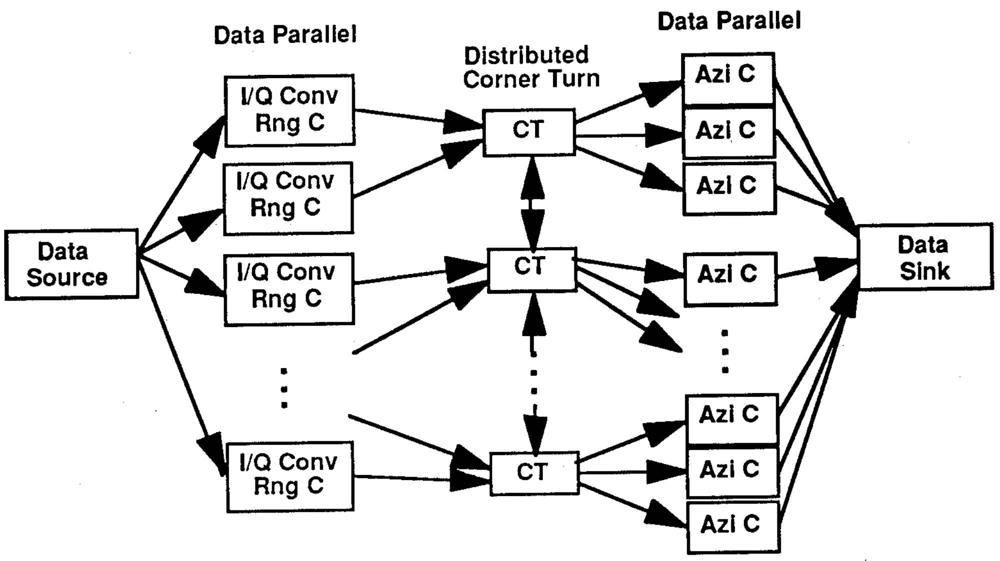


Figure 4-6. Scalable Real-Time SAR Processing

SECTION 5

CONCLUSION

A parallel software design process for real-time embedded applications has been described. A test bench implemented on the target architecture is used to assess the level of real-time performance achieved by any subsection of the system (single node application kernels building up to the completely integrated system). The simple control and data flow used in the test bench approach eases the integration process and can provide the basis for data flow shell synthesis tools.

The design process was effectively applied in mapping the RASSP SAR benchmark onto an Intel Paragon. A single polarization solution was demonstrated with 12 (GP) nodes, and projections to the Embedded Touchstone suggest that a 14 (MP) node system will support processing of all three polarizations including I/O. These solutions use the current non real-time OSF/1 operating system but still provide worst-case timing guarantees that meet the RASSP real-time constraints. Fully taking advantage of the higher performance SUNMOS operating system would reduce this node count, as would presumably a future real-time version of OSF/1. Future work will focus on the area of real-time scalability. A scalable corner turn implementation is critical for truly scalable signal processing and to facilitate application of software synthesis tools.

LIST OF REFERENCES

Brown, C. P., M. I. Flanzbaum, R. A. Games, and J. D. Ramsdell, 1994, *Real-Time Embedded High Performance Computing: Application Benchmarks,* MTR 94B145, The MITRE Corporation, Bedford, MA.

Brown, C. P., R. A. Games, and J. J. Vaccaro, 1995, "Implementation of the RASSP SAR Benchmark on the Intel Paragon," *Proceedings of the 2nd Annual RASSP Conference*, ARPA, pp. 191–195.

Blitzer, F., 1993, "Militarized Touchstone Program," *Proceedings of the 1993 IEEE National Aerospace and Electronics Conference*, Vol. 1, Dayton, OH, pp. 137–143.

Gustafson, J. L., 1988, "Reevaluating Amdahl's Law," *Communications of the ACM*, Vol. 31, No. 5, pp. 532–533.

Maccabe, B., K. S. McCurley, and R. Rissen, November 1993, *SUNMOS for Intel Paragon: A Brief User Guide*, ftp://ftp.cs.sandia/pub/sunmos/papers/ISUG94-1.ps.Z, Sandia National Laboratories, Albuquerque, NM.

McCurley, K. S., November 1993, *Intel NX Compatibility under SUNMOS*, Technical Report No. SAND 93-2618, Sandia National Laboratories, Albuquerque, NM.

Oppenheim, A. V., and R.W. Schafer, 1975, *Digital Signal Processing,* Englewood Cliffs, N.J.: Prentice-Hall, Inc.

Zajcew, R., P. Roy, D. Black, C. Peak, P. Guedes, B. Kemp, J. LoVerso, M. Leibensperger, M. Barnett, F. Rabii, D. Netterwala, January 1993, "An OSF/1 Unix for Massively Parallel Multicomputers," *Proceedings of the 1993 Winter USENIX Conference*, San Diego, CA, pp. 449–468.

Zuerndoerfer, B., and G. A. Shaw, 1994, "SAR Processing for RASSP Application," *Proceedings of the 1st Annual RASSP Conference*, ARPA, pp. 253–268.

☆U.S. GOVERNMENT PRINTING OFFICE: 1996-710-126-20092